Express Mail No. EL821158045US
Docket No. STL920000096US1
Firm No. 0055.0041

METHOD, SYSTEM, AND PROGRAM FOR TRANSFERRING DATA FROM AN APPLICATION ENGINE

RELATED APPLICATIONS

[0001] This application is related to the following copending and commonly assigned patent applications, which are incorporated herein by reference in their entirety:

"Method, System, and Program for Generating a Workflow", having attorney docket no. STL920000094US1 and filed on June 28, 2001;

"Method, System, and Program for Using Objects In Data Stores During Execution of a Workflow", having attorney docket no. STL920000095US1 and filed on June 28, 2001;

"Method, System, and Program for Executing a Workflow", having attorney docket no. STL920000099US1 and filed on June 28, 2001; and

"Method, System, And Program For Enabling Access to a Plurality of Services", having attorney docket no. STL920000090US1 and filed on the same date herewith.

BACKGROUND OF THE INVENTION

1. Field of the Invention

[0002] The present invention relates to a method, system, and program for transferring data from an application engine.

2. Description of the Related Art

[0003] A workflow program allows businesses and other organizations to define their business operations as a computer model known as a workflow. A workflow defines a series of processes to be performed by users at a client computer. The user activities at the client computers may involve updating an electronic form, reviewing information, etc. After one user in the workflow performs a specified action, the work item or other

information is then routed to one or more further nodes where further action may be taken. For instance, an on-line purchase of a product may involve numerous steps, such as receiving the customer order, routing the customer order to the credit department to process the bill and then routing the order to the shipment department to prepare the shipment. Once the shipment is prepared, the product may be shipped and information on the purchase is then transferred to the customer service department to take any further action. Each of these processes may be defined as nodes in a workflow. A workflow program would then route the customer order to the business agents designated to handle the job. For instance, the initial order would be received by the order department and then routed to a person in shipping and billing. Once the bill and package are prepared, a further invoice may be forwarded to shipping. After shipping sends the package, the shipping agent may then enter information into the invoice and forward the electronic invoice to customer service for any follow up action.

[0004] A workflow is designed using workflow software, such as the International Business Machines (IBM) MQSeries Workflow software product. Often application programs that interface with a workflow engine require information on workflows operating at the workflow engine. The process of transferring information from the workflow engine to an application over a network can take a considerable amount of time and burden network resources given the large size of data used to describe workflows available at a workflow engine.

[0005] For these reasons, there is a need in the art to provide improved techniques for providing information on workflow and other services available in a network environment to application programs.

SUMMARY OF THE PREFERRED EMBODIMENTS

[0006] Provided is a method, system, and program for enabling access to resource objects in an application engine. A request is received from a calling entity for resource objects of a specified type in the application engine. A request to the application engine

is generated for information on available resource objects of the specified type. In response to receiving the information from the application engine, a collection object is generated including one metadata element for each resource object of the specified type in the application engine. The generated collection object is returned to the calling entity.

[0007] In further implementations, the application engine is one of a plurality of service engines enabling access to service resources. The request for the resource objects from the calling entity comprises a method that is a member of a service class implementation of the application engine, wherein each service engine provides one service class implementation of methods and objects from a same abstract service class.

[0008] Still further, the collection object is generated using methods from a collection object class. A retrieve method is received in the collection object class from the calling entity requesting the resource object represented by one selected metadata element in the collection object. An additional request to the application engine is generated for the resource object requested in the retrieve method. The requested resource object is received from the application engine and the requested resource object is returned to the calling entity invoking the retrieve method.

[0009] The described implementations provide a technique to allow an application or user to receive information on resources available at an application engine and retrieve the resource objects for certain of the resources identified in the information. With the described implementations, information on resources available in the application engine and the resources themselves are provided as needed.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 illustrates a workflow computing environment in which aspects of the invention are implemented;

FIG. 2 illustrates logic performed by a workflow server to execute a workflow in accordance with implementations of the invention.

FIG. 3 illustrates an architecture of object oriented classes for implementing a workflow in accordance with implementations of the invention; and

FIGs. 4 and 5 illustrate logic to utilize the methods and objects from the object oriented class architecture of FIG. 3 to execute a workflow in accordance with implementations of the invention;

FIG. 6 illustrates an architecture of object oriented classes for implementing workflow services for heterogeneous services in accordance with implementations of the invention;

FIG. 7 illustrates an implementation an environment for enabling access to heterogeneous services in accordance with implementations of the invention;

FIG. 8 illustrates an example of a service object maintaining information on services in accordance with implementations of the invention;

FIG. 9 illustrates logic to construct a service in accordance with implementations of the invention; and

FIG. 10 illustrates logic to construct a connection object in accordance with implementations of the invention;

FIG. 11 illustrates logic to obtain information on a named service in accordance with implementations of the invention;

FIG. 12 illustrates an example of a collection object including metadata in accordance with implementations of the invention;

FIG. 13 illustrates logic to provide information on resources for an engine in accordance with implementations of the invention; and

FIG. 14 illustrates logic to access resource objects for which metadata information is included in the collection object.

Express Mail No. EL821158045US
Docket No. STL920000096US1
Firm No. 0055.0041

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0011] In the following description, reference is made to the accompanying drawings which form a part hereof and which illustrate several embodiments of the present invention. It is understood that other embodiments may be utilized and structural and operational changes may be made without departing from the scope of the present invention.

[0012] FIG. 1 illustrates a workflow environment implementation in which the invention is realized. A workflow engine 2 includes a runtime database 4 and a workflow server 6, such as the IBM MQSeries Workflow server. The workflow server 6 is capable of transforming a workflow model coded in a workflow definition language (WDL) file 10, such as FDL, into a process template 8 implemented in the runtime database 4. The runtime database 4 stores database tables that implement the data structures that provide the status and setup information needed for workflow process execution. Whenever the state of a process activity changes, such information is recorded in the runtime database 4. The runtime database 4 may be implemented using any database program known in the art, such as IBM DB2.**

[0013] The workflow server 6 coordinates and manages the execution of processes for a defined process template 8. The workflow server 6 executes any programs associated with a process defined for the workflow, interprets the process definitions, creates process instances and manages their execution, manages processes and states, logs events, communicates with users as part of the workflow, etc. The workflow server 6 may include a database client program (not shown) to access and update records related to the workflow being processed maintained in the runtime database 4. The workflow server 6 processing may be distributed across multiple computers to achieve workload balancing.

[0014] The workflow clients 12a, b...n represent the client computers that execute workflow application program interfaces (APIs) to perform workflow related actions and activities and return messages to the workflow server 6. The workflow clients 12a, b...n thus comprise instances of the workflow code on the client computers that allow users to

interface with the executing workflow and the workflow server 6. The workflow server 6 would execute activity programs as part of executing the workflow and transmit messages and data to the workflow client 12 to request user action to proceed with the workflow. The actions associated with the nodes and executed by the workflow server 6 may comprise Java servlets. The workflow client 12 may comprise a Web browser capable of executing Java scripts transferred from the Java servlet executing on the workflow server 6. Further, details on implementations and interactions of the workflow server 6 and client 12 are described in the IBM publication "IBM MQSeries Workflow: Concepts and Architecture, Version 3.3", IBM document no. GH12-6285-03 (March, 2001), which publication is incorporated herein by reference in its entirety.

[0015] A workflow builder 20 comprises a system including a buildtime program 22 that implements a plurality of graphical user interface (GUI) panels in which a user may define the components of a workflow model 24. A workflow translator 26 converts the workflow model 24, with the defined workflow components, into a workflow definition language (WDL) file 10 that implements the workflow model 24. The workflow definition language (WDL) may comprise the FlowMark Definition Language (FDL), Workflow Process Definition Language (WPDL) or any other workflow definition language known in the art that is used to define workflows. The workflow translator 24 would transfer the WDL file 10 to the workflow server 6 to transform into a process template 8 in the runtime database 4 in a manner known in the art. Further details of using the buildtime program 22 to build workflows are described in the copending and commonly assigned patent application "Method, System, and Program for Generating a Workflow", having docket no. STL920000044US1, which application was incorporated herein by reference above.

[0016] The workflow engine 2, and each of the program components therein, such as the runtime database 4 and workflow server 6, may be implemented in one or more computing machines. The workflow clients 12 which provide the workflow interface to users may be implemented on one or more client machines. The workflow builder 20,

including the buildtime program 22 and workflow translator 26 programs, may be implemented on one or more computing machines. Any portion of the workflow engine 2, workflow builder 20, and/or workflow client 12, and program components therein, may be implemented on the same computing machines or separate machines. The computing machines used to implement the workflow engine 2, workflow clients 12, and workflow builder 20 may comprise any computing device known in the art, such as a server, workstation, mainframe, personal computer, laptop computer, hand held computer, telephony device, etc.

[0017] One use of a workflow is to generate a final product, which may comprise the result of the effort of a single business unit or the cumulative efforts of multiple users and units within an organization. To produce the final product, a workflow packet comprised of one or more documents would transfer through various user work stations in the company defined as nodes in the workflow to require the user associated with such node to handle and process and forward to another user to handle. A document is comprised of a multimedia item that has digital content.

[0018] For instance, an insurance company may have to process numerous documents related to an insurance claim, such as photographs, appraisals, expert reports, etc. Employees may spend a substantial amount of time sorting through documents and associating the documents with particular claims. In the workflow model, all the documents related to a single claim would be part of a work packet that may move through various user stations to review and process. The workflow would comprise the flow of work and actions that are performed on the documents or workflow packet by multiple users in the system.

[0019] The workflow defines the sequence and boundaries of how the work is performed with respect to the documents in the workflow packet, and any restrictions on the order in which documents in the workflow packet must be processed. For instance, before the claim can proceed to a further step, a claims adjuster might be required to ensure that certain documents are included in the workflow packet for the claim before

the workflow packet can proceed to further nodes in the workflow, e.g., determining the amount of compensation.

[0020] In workflow terminology, a worklist is a queue of work items. Each work item comprises a unit of work for a node in the workflow that is performed by the users associated with that node. Each work item may be associated with one work packet, which comprises documents or objects that are processed during the work defined for that work item. When a user at one node accesses the work item to perform the work defined therein, that work item is locked, thereby preventing others at that node from accessing the work item.

[0021] A worklist, which is a queue of work for the users of the organization to perform with respect to the workflow packet. The work items within the worklist can be handled by any of the employees/users assigned to the worklist. An action list defines the actions that a user can perform on the work packet objects associated with the work item, such as selections or data that may be entered in the work packet. For example, an adjuster in the claim process workflow can select an option to continue consideration of the claim if it appears valid or select an option to reject the claim. The workflow further consists of the paths defined as the connections between nodes which indicate the order of execution of nodes in the workflow.

[0022] An action list may be associated with a workflow that provides a list the actions that can be invoked at the nodes in the defined workflow. The actions may comprise programs that are executed at a particular node. In certain implementations, the actions comprise Java methods that the workflow server 6 executes when control proceeds to the node with which the method is associated. Action in the list would be associated with particular nodes. An access list defines a mapping of users that can be assigned to nodes to perform the action associated with such node. An notification feature causes a message to be sent to a specified user if the user associated with a node has not performed the action defined for the node within a specified time frame.

Express Mail No. EL821158045US
Docket No. STL920000096US1
Firm No. 0055.0041

[0023] One or more actions and a user with are associated with the work nodes in the workflow. The work nodes defined for the workflow may comprise a decision point node, collection point node, document node, and assign value node. A decision point node causes the workflow to proceed along a branch of execution based on selection by the user or some other action taken by an external application called at a previous work node. For instance, the path taken to the next node in the workflow may vary if the claim adjuster selects to reject the claim as opposed to approving the claim. A collection point node is a work node where certain documentation is gathered and added to the work packet. The collection node holds and manages work packages that cannot be processed completely until additional information is received. A document node represents a document in the workflow.

[0024] In certain implementations, the workflow model 24 defined using the buildtime program 22 is document centric in that the actions performed at the node concern the processing of work packages that may comprise any content or object that is processed and routed through the workflow. FIG. 2 illustrates the logic performed by the workflow server 6 to execute a workflow. When a user invokes a workflow stored in the runtime database 4, the workflow server 6 accesses (at block 100) the start node of the invoked workflow by interacting with the runtime database 4 in a manner known in the art. From the properties defined for that node, the workflow server 6 determines (at block 102) the actions and user associated with the node. The workflow server 6 further processes (at block 104) the access list defined for the workflow to determine the work item for the accessed node. If (at block 106) the determined work item currently accessed in the workflow is locked by another user at that node, then the workflow server 6 waits (at block 108) for the lock on the work item(s) to be released. If the work item is not locked or after the lock is released, control proceeds to block 110 where the workflow server 6 places a lock on the determined work item. The workflow server 6 then executes (at block 112) the action associated with the node and communicates data to the workflow client 12 of the determined user requesting user action.

[0025] If (at block 114) notification is enabled for the current node and the deadline has passed (at block 116) without receiving a response from the user, then the workflow server 6 notifies the user specified with the enable notification that the deadline has passed. Upon receiving (at block 118) a response from the user, which may comprise entering information, modifying a work item, adding a work item to the work package, selecting an option, etc., the workflow server 6 unlocks (at block 120) the work item(s) previously locked for the user. If (at block 122) the current node is the stop node, then control ends; otherwise, if there are further nodes to process in the workflow, then the workflow server 6 determines (at block 124) from the path from the current node the next node in the workflow and accesses (at block 126) the next node. Control then proceeds back to block 326 to process the next node.

[0026] The workflow logic of FIG. 2 provides a document centric workflow in that the state of processing work items associated with the node controls the workflow because control cannot proceed to other subsequent nodes that process the locked work item until the node holding the lock completes execution and releases the lock on the work item. Thus, access to work items controls the flow through the workflow.

[0027] With the described implementations, the workflow builder 20 generates a WDL file 10 that may be compatible with workflow engines from different vendors because different vendors may design their workflow engines to be compatible with the WDL format of the WDL file 10. This allows the workflow model defined in the WDL file 10 to be transportable across different vendor workflow engine platforms.

Object Oriented Workflow Architecture

[0028] FIG. 3 illustrates an architecture of object oriented classes and their interrelationship that are used to implement a workflow of nodes. As indicated in the legend 400, a rectangle indicates a class; a line connecting classes indicates an association of the connected classes; a line connecting classes terminating in a filled circle indicates that there may be one or more instances of the class at the end with the circle for each

instance of the class at the other end of the line; and a line terminating at a diamond indicates that the class at the diamond end is an aggregate, such that the aggregate object is made up of one or more instances of the class at the other end of the line. FIG. 3 illustrates the relationship of the classes.

[0029] The WorkFlowService class 402 is the starting point for a user wanting to access a workflow. The WorkFlowService class 402 includes methods that allow users to access already defined workflow templates and executing workflows. The WorkFlowService class 402 is associated with the WorkFlowTemplate 404, WorkFlow 406, and WorkFlowList 408 classes. The WorkFlowTemplate class 404 provides methods that allow the user to manipulate workflow process template objects, e.g., process template 8 (FIG. 1), which comprise a defined workflow that is stored in the workflow engine 2. The WorkFlow class 406 provides methods that allow the user to access information and control an executing workflow. The WorkList class 408 includes methods that allow the user to access an executing work list object comprised of work items and information on the current state of the executing work list, i.e., information on work items being processed. The methods in the WorkFlowService class 402 are used to retrieve information on particular workflows, workflow templates, and workflow lists associated with a particular workflow service. The methods from the other classes, such as the WorkFlowTemplate 404, WorkFlow 406, and WorkFlowList 408 classes, can then be used to obtain specific information and control over those workflow templates, workflows, and workflow lists identified by the WorkFlowService class 402 methods.

[0030] The WorkFlowTemplate class 404 provides information on a workflow template. A workflow object from the WorkFlow class 406 represents an executing workflow. The WorkFlowContainer class 410 includes methods to instantiate a container object that includes information on one container used to transfer data between nodes. Users at nodes may access data in the container and update the container with additional data. The data in the container may be used by the action being executed at a node. The WorkFlow class 406 is associated with the WorkFlowNotification class 412,

which is used to provide notifications, such as notifications if a user does not perform an action at a node within a predefined time period. There may be many notifications provided for one workflow. The WorkFlow class 406 is further associated with the WorkFlowItem class 414, such that one executing workflow may be associated with one or more work items indicating a unit of work to perform for a node within the workflow. The WorkFlowItem class 414 is associated with the WorkFlowContainer class 410, such that one container may be used at a work item to provide data to the user executing the unit of work defined by the work item. The relationship between the WorkFlow class 406 and the WorkFlowItem class 414 indicates that there may be many work item objects associated with one executing workflow. The class architecture of FIG. 10 further illustrates that a workflow list of the WorkFlowList class 408 is an aggregate of the workflow from the WorkFlow 414 Item class and workflow notifications from the WorkFlowNotification 412 class.

[0031] The above object oriented architecture of FIG. 10 defines how the different classes interrelate in order to implement a workflow. Each of the above interrelated classes 402, 404, 406, 408, 410, 412, and 414 provides interfaces/methods that may be used within a workflow computer program to implement the workflow and actions performed at a node. The workflow program would be executed by the workflow server 6 (FIG. 1) in the workflow engine 2.

[0032] Following are examples of some methods of the WorkFlowService class 402, including:

WorkFlowService(): constructs a new workflow service, which provides access to different workflow services in the workflow engine 2 (FIG. 1). Each workflow service is associated with workflow templates, executing workflows, and workflow lists of work items for a workflow.

connect: provides a user name, authentication, and connection string to use to authenticate a user to provide access to a requested workflow service, which allows access to workflow templates, work lists, etc.

connection: handle returned to a user to allow access to a particular workflow service.

setDatastore: a reference to a data store including documents and objects used by the work items in the workflows associated with the workflow service. Thus, different workflows for a workflow service may process documents within workflow packages from the same data store.

listWorkFlows: returns a list of all workflow objects of the WorkFlow class 406.

listWorkLists: returns a list of all work list objects of the WorkFlowList class 408.

listWorkFlowTemplates: returns a list of all template objects of the WorkFlowTemplate class 404.

[0033] Following are examples of some methods of the WorkFlowService class 402, including:

WorkFlowTemplate(): constructs a workflow template object including a defined workflow. This workflow template may be created using the GUI panels and buildtime program described above.

name: returns name of a workflow template.

description: returns a description of the work performed by a workflow template.

modifiedTime: time the workflow template was last modified.

[0034] Following are examples of some methods of the WorkFlow class 406, including:

WorkFlow(): constructs a workflow object representing a workflow comprised of nodes and work items for a specified workflow. The workflow may also be provided a container that is used to allow users of different work items to communicate and/or a work packet comprised of one or more documents or objects to be processed as part of the workflow.

get/setName: returns or sets the name for a workflow.

workFlowTemplateName: returns the name of the workflow template associated with the workflow.

notificationTime: returns the time of the last notification generated for the workflow in response to a user not performing an action for one accessed node within a specified time period.

modifiedTime: Returns the last time the workflow was modified.

stateChangeTime: returns the last time a state change occurred with the workflow:

startTime: returns the time the workflow was started.

endTime: returns the time the workflow ended.

state: returns a state of the workflow, such as ready, running, finished, terminated, suspended, terminating, suspending, deleted, etc.

inContainer: returns the input container associated with the workflow.

start: starts a workflow with a container if the state is ready.

terminate: terminates the workflow if the state is running, suspended, or suspending.

suspend: suspends the workflow if the state is running.

resume: resumes a suspended workflow if the state is suspended and suspending.

add: adds a workflow to the system that is associated with one specified workflow template.

[0035] Following are examples of methods of the WorkFlowContainer class 410, which instantiates a container object used with a workflow to transport information among the nodes.

WorkFlowContainer(): constructs a container object for a container used within a particular workflow.

get/setPriority: get/sets the priority for an item in the container.

get/setActivityNode: get/sets the current node being processed, may also get/set information on the current activity node.

get/setWorkPacketID: get/sets an identifier of a work packet being routed through the system.

get/setActionPerformed: get/sets information on an action being performed.

get/setUserVariable: get/sets a variable maintained in the container, that may have predefined values. The priority is maintained for a user variable in the container.

retrieve: retrieves and refreshes the container.

update: updates the container data.

[0036] Following are examples of some methods of the WorkList class 408, where a work list object is a representation of a work list in the system. As discussed, a work list object comprises a collection of work items and notifications for an executing workflow.

WorkList(): constructs a work list object for a specified work list. A work list consists of work items.

get/set ACLName: get/sets the action control list (ACL) name for the work list including the actions that may be performed as part of units of work for the work list.

listWorkItems: lists the work items on the work list.

listWorkItemsByTemplate: returns the work items for the work list by the specified workflow template name.

listWorkItemsByNode: returns a list of the work items assigned to each node in the work flow.

listProcessNotifications: lists notifications generated during workflow that are associated with the workflow process. For instance, the notification enabled through the GUI in FIG. 3 provides a general notification for the workflow. In

certain implementations, a notification process is activated and performed as a background process to generate notifications.

listActivityNotifications: lists notifications generated during workflow that are associated with a particular activity, such as a user not performing an activity within a specified time. For instance, the notification enabled through the GUI of FIGs. 4 and 5 enables notifications for activities at particular nodes.

add/update/delete/retrieve: separate commands that allow user to add, update, delete, and retrieve a work list.

[0037] Additional commands may be provided to access the information in the work list, such as filter commands to provide filters for accessing information from the work list, thresholds of the number of items that can be in the work list, etc.

[0038] Following are examples of some methods of the WorkFlowItem class 414, where a work item object represents a unit of work performed in the workflow. The following methods are used to create and modify work items, and obtain information thereon.

WorkFlowItem(): constructs a work item for a specified workflow, node, and owner.

name: returns the name of the node to which the work item is assigned.

state: returns a state of the work item, such as not set, ready, running, finished, terminated, suspended, disabled, checked out, in error, executed, etc. A work item is checked out when a user has accessed the work item to perform the actions defined for the work item.

workFlowName: returns the name of the workflow including the work item.

workFlowTemplateName: returns the name of the workflow template including the work item.

priority, owner, notificationTime, startTime, creationTime, modifiedTime: methods that return information on the priority, owner, time of last notification, time of creation and time of last modification for a work item, respectively.

retrieve, start, finish: methods used to retrieve, begin executing, and complete a work item, respectively.

checkIn, checkOut: checkOut locks a work item to prevent other users at a node from accessing the work item and changes the state of the work item to checked out. Upon check out, the container associated with the work item is accessed from the previous node using the inContainer method. The checkIn method receives the completed work item from the user, releases the lock, and provides the container to route to the next node.

inContainer: method that obtains container from previous node for use with work item checked out at current node being processed.

outContainer: method generates an out container to include contents of container user accessed at work item, including any changes made by the user to the data in the container. A handle of the out container is generated and provided with checkOut method called for the next node to provide that container to the user of the next node in the workflow.

[0039] Following are examples of some methods of the WorkFlowNotification class 412, where a notification object represents a generated notification. The following methods are used to create and modify notifications, and obtain information thereon.

WorkFlowNotification(): constructs a notification object having a specified notification name, notification type, and owner name for a specified workflow service and workflow. The notification type indicates how the owner is notified.

state: returns a state of the notification, such as not set, ready, running, finished, terminated, suspended, disabled, etc.

priority, owner, notificationTime, startTime, creationTime, modifiedTIme, receivedTime: these methods return the priority of the notification, owner of the notification, time that must elapse before the notification is generated, time the notification started, time the notification was crated, time of last notification to the notification, time the notification was received, respectively. The notification would be started and executed as a background process.

receiveReason: returns a received reason for the notification.

retrieve, cancel: methods that retrieve and cancel a notification, respectively.

transfer: transfers a notification to a specified user. In this way, a notification can be transferred from the current owner to some other user.

[0040] The above described methods and classes would be included in a workflow program executed by the workflow server 6 (FIG. 1) to execute the workflow. The methods described above would be used to access and modify the workflow related objects, such as the workflow, work items, notifications, containers, etc. when running the workflow. The above described methods may also be used in other programs that can obtain information and status on a workflow.

[0041] FIGs. 4-5 illustrate an example of program logic in a workflow program executed by the workflow server 6 (FIG. 1) utilizing the above discussed methods to implement a workflow. With respect to FIG. 11, control begins at block 450 where the program calls the constructor methods, WorkFlowService() to construct a workflow service object. The workflow program would then call (at block 452) the WorkFlowService list methods, such as listWorkFlows, listWorkLists, listWorkFlowTemplates, to obtain information on the workflows, workflow templates, and work lists for a workflow service. This information may then be presented to a user for selection. Various other methods in the classes may be called to access information on the workflow to present to the user when making a decision on which workflow to execute.

[0042] At block 454, user selection of a workflow to process is received. The workflow program then calls (at block 456) the WorkFlow start method to start the workflow. The workflow program then calls (at block 458) the listWorkItemsByNode method to obtain all the work items for the started workflow, and the nodes to which the one or more items are associated. The workflow program then performs a loop at blocks 460 through 490 for each node *i* in the workflow, as determined from the list of work items by node. For each node *i*, the workflow program performs a loop at block 462 to 488 for each work item *j* associated with node *i*. If (at block 464) there is a notification for the work item and the user that is the owner of the item, as determined from the methods, then the workflow program retrieves (at block 466) retrieves the notification and then starts a monitor to determine if the time period for the notification has elapsed without the work item completing. From block 464 or 466, the workflow program calls (at block 468) the checkOut method to lock the work item *j*. The inContainer method is called (at block 470) to access any container associated with the work item *j*. Once the work item *j* is locked, the workflow program then executes (at block 474) the actions associated with the work item *j*.

[0043] Control then proceeds to block 476 in FIG. 5, where the workflow program calls container get and set methods to access or modify the data and variables in the container accessed for the work item *j* in response to executing actions assigned to that work item *j*. For instance, as part of performing actions for a work item, the user of the work item may read and write data to the container. The workflow program receives (at block 482) indication from a user that the actions associated with the work item have completed. The workflow program further calls (at block 486) the checkIn method to release the lock on the work item *j* and the outContainer method to generate a new container including any updates to provide to the user at the next node in the workflow. The handle to the new container would be used in the next called checkOut method to provide the container to the user at the next node of the workflow. If there are further work items for the node *i*, then control proceeds (at block 488) back to block 452 to retrieve the next work item.

After completing all the work items for node *i*, control proceeds (at block 490) back to block 460 to process the next node in the work list.

[0044] The above described logic utilized workflow related classes and the methods therein to implement a workflow and obtain information thereon. The workflow server 6, or some other component in the workflow engine 2 (FIG. 1), would then translate the workflow objects and methods into application specific commands, such as Structured Query Language (SQL) commands to manipulate the data in the runtime database 4 and process template 8 to obtain information on the workflow and implement workflow operations.

Enabling Access to Heterogeneous Services

[0045] The above described architecture described how programmers in a client/server worfklow environment can write application programs to perform worfklow related operations using an object oriented class architecture. Further implementations provide methodologies for allowing a client application program to utilize heterogeneous services, e.g., a workflow program, search engines, a data repository and program for accessing a data repository (such as the IBM Content Server*) etc., from different vendors. A service is a set of functionality or program that may be accessed and executed, such as a workflow program.

[0046] FIG. 6 illustrates an architecture of a client/service server framework to provide access to different types of service programs from different vendors. The architecture provides an abstract service class 500 that implements different abstract service types, a workFlowService class 502 and another service class type 504. The general abstract service class 500 provides methods and objects that all services, e.g., workflow engine vendors, search engine vendors, database program vendors, etc., must implement to make their services available to users, such as client and application programs, of the client/server architecture described herein. The abstract service type class 502 and 504 provides methods and objects that all services of a particular type must implement such as

the methods and objects that all workflows must implement, all search engines must implement, etc. The WorkFlowService server 506 and client 508 side classes provide the methods and objects for one vendor implementation of a workflow service and OtherService client 510 and server 512 classes provide the methods and objects for one particular implementation of another type of service, e.g., a search engine, etc.

[0047] In FIG. 6, a line connecting two classes 500, 502, 504, 506, 508, 510, 512 represents an association between the two classes connected by the line, the triangle arrow on one end of the line indicates that the class at the other end of the line inherits the attributes of the class pointed to by the arrow. Thus, abstract workFlowService class 502 inherits the attributes of the abstract service class 500 and the WorkFlowService classes 506 and 508 inherit the attributes of the abstract workFlowService class 502, which would include the parent service class 500.

[0048] FIG. 7 illustrates further details of an implementation of the client/server architecture for enabling access to different services from different vendors using the class architecture described with respect to FIG. 6. A plurality of workflow clients 550a, b, c comprise client computers, such as workstations, laptops, or any other computing device known in the art. Each workflow client 550a, b, c would include an instance of a workflow service object 552a, b, c (only 552a in client 550a is shown) instantiated from the WorkFlowService (client side) class 506. Each workflow client 550a, b, c includes a communication protocol 554a, b, c (only communication protocol 554a in client 550a is shown) that enables communication between the workflow client 550a, b, c and a workflow server 556. The communication protocol 554a, b, c comprises a network communication protocol known in the art, such as the Remote Method Invocation (RMI), CORBA, DCOM, etc., is used to enable communication between methods and objects in the WorkFlowService client 556 to the server side 558. The workflow clients 550a, b, c would communicate with the workflow server 556 over a network 564.

[0049] The clients 550a, b, c further include a translator to execute methods in the WorkFlowService (client side) 506 class. For instance, if the WorkFlowService (client

Express Mail No. EL821158045US
Docket No. STL920000096US1
Firm No. 0055.0041

side) 506 class is implemented in an object oriented language, such as Java**, then the client 550a, b, c would include a translator, such as a Java Virtual Machine (JVM), to translate the method statements to executable code.

[0050] The workflow server 556 includes a workflow service object (server side) 558 that maintains information on a particular vendor implementation of a workflow engine 560. The workflow server 556 may comprise one or more server systems, workstations, or any other computing device known in the art. The workflow service object (server side) 558 is instantiated using classes and methods from the WorkFlowService (server side) class 508 for the particular workflow engine 560. In certain implementations, a user or workflow application 562 could execute methods from the WorkFlowService (client side) 506 class at the workflow client 550a, b, c to access information from the workflow service object (client side) 552 or from the workflow engine 560. In response, such methods would be communicated via the communication protocol 554a, b, c to the workflow server 556 for execution as methods in the WorkFlowService (server side) class 508. Such client methods may be executed against the workflow service object 558 for information maintained in the workflow service object 558, such as information on the type or category of the workflow engine 560 and a connection, or executed against the workflow engine 560.

[0051] In the described implementations, the WorkFlowService (client side) 506 and WorkFlowService (server side) 508 implement the same methods, with the exception that methods called on the client side are transferred via the communication protocol 554a, b, c, 574a, b, c to the server 556, 576 for execution thereon.

[0052] Methods invoked on the workflow server 556 from the WorkFlowService object (server side 558) are translated by a program interface 564 to the native code used by the workflow engine 560. The program interface 564 for translating methods from the WorkFlowService (server side) class 508 to native code may comprise a Java Native Interface (JNI), or any other program known in the art for providing a translation from one language to native code used by the target program or service.

[0053] Alternatively, method calls may be initially made on the workflow server 556, thereby avoiding the need for communication through the communication protocol 554.

[0054] A workflow application 562 is written in a computer language, such as an object oriented language, e.g., Java, C++, etc., and includes code to implement a workflow using methods and objects from the WorkFlowService classes 506, 508.

[0055] In the described implementations, each workflow vendor wanting to make their workflow engine 560 product available to users of the architecture would have to develop methods and classes implementing an instance of the WorkFlowService (client side) 506 and (server side) 508 classes, which implement the abstract workFlowService class 502, including at least the methods and objects of the abstract workFlowService 502 and service 500 classes. In certain implementations, the vendors would utilize the same programming language to provide their WorkFlowService 550a, b, c implementation to users of the architecture, such as the Java programming language**. The program interface 564 would then translate the vendor implementation of the WorkFlowService 550a, b, c to the native code of their workflow service 556.

[0056] Similarly, other service clients 570a, b, c comprise computers including an other service object (client side) 572a, b, c instantiated from the OtherService (client side) class 510. Calls at the client side would be communicated via a communication protocol 574 to one other service server 576, including an other service object (server side) 578 instantiated from the OtherService (server side) class 512. An interface 580 translates the methods from the OtherService (server side) class 510 to the native code of the other service engine 582, such as a search engine or any other server-type application program known in the art. An other service application 584 includes methods from the OtherService (client side) 510 or (server side) 512 classes to obtain information from the other service objects (client side) 572 or (server side) 578) or the other service engine 582 in the same manner described with respect to the workflow service implementation.

[0057] Moreover, one client may utilize the methods from multiple client side service classes, e.g., from the WorkFlowService (client side) class 506 and the OtherService

(client side) class 510 to access both service engines, e.g., one workflow service engine and one search engine, from a same client. Still further one application program may include methods to access different services.

[0058] Thus, the abstract classes 500, 502, and 504 provide the methods and objects that any service participating in the heterogeneous service environment must implement.

[0059] The service objects (client side) 558 and 578 include information about one connection and the service, e.g., workflow engine 560 or other service engine 582. In certain implementations, one workflow service object (client side 552 and server side 558) or other service object (client side 572 or server side 578) is maintained for each active and authenticated connection to the workflow engine 560 and other service engine 582.

[0060] In certain implementations, all of the programs and objects shown in FIG. 7 are implemented in a distributed computing system including numerous connected computing machines.

[0061] Following are examples of some methods and objects in the abstract service class 500, including:

> connect(): this method takes as parameters a service name, user name, authentication, and an optional connection string. The method authenticates the user having the user name and authentication with the named service. If the user authenticates, then connection information is generated and the handle is returned that addresses the authentication information for the user. The connection information would be stored in one service object 558, 578. The user would use the handle to the connection information when requesting information from the engine 560, 582. The targeted engine 560, 582 would then use the handle to verify the user's authentication information to determine if the user is authorized access to the requested resource provided by the engine 560, 582. The handle may subsequently be used to allow the user to access authentication information

without having to go through the authentication process each time the user requests resources from the engine 560, 582.

disconnect(): voids the authentication so that the handle that was provided no longer enables access to the engine 560, 582.

serviceName(): returns the name of a service at the engine 560, 582.

serviceType(): returns information on a type of engine 560, 582, e.g., workflow, search engine, etc., by the vendor providing the service.

serviceCategory(): returns information distinguishing the category of a engine 560, 582, e.g., workflow, search engines, etc.

userName(): returns the user name associated with a connection handle.

isConnected(): returns a boolean value indicating the connection status for a connection handle.

[0062] Following are examples of some methods and objects in the abstract workFlowService 502, including:

getDatastore(): returns a reference to a data store 554 associated with a service defined for the engine 560, 582.

listWorkFlows(): returns a list of workflow objects, such as workflow objects created using the WorkFlow class 406 methods discussed above with respect to FIG. 3.

listWorkTemplates(): returns a list of workflow template objects, such as workflow template objects created using the WorkFlowTemplate method discussed above as part of the WorkFlowService class 402.

listWorkLists(): returns a list of work list objects, such as work list objects created using the WorkList class 408 discussed above with respect to FIG. 3.

[0063] The vendor implemented WorkFlowService classes 506 and 508 includes the methods and objects discussed with respect to the WorkFlowService class 402 discussed

above and also includes the methods and objects discussed above with respect to the abstract service 500 and workFlowService classes 502. Similarly the vendor implemented OtherService 510, 512 classes would implement the abstract class 500 as well as methods and objects needed to enable access to information in the service objects providing information on one engine 560, 582. c. For instance, the OtherService 510, 512 classes would include methods to connect to the other service and access the resources of the engines 560, 582.

[0064] Following are examples of some methods and objects in the implementation of the WorkFlowService classes 506 and 508, including:

WorkFlowService(): constructs a workflow service object 552a, b, c, 558. Each workflow service object includes information on groupings of resources available at one workflow engine 560, e.g., a name of the service, category, type, connection information, etc.

connect(): constructs a connection handle object including the authentication information a user needs to connect to one service engine 560, 582. A connection object would be stored within one of the workflow service objects (server side) 558.

[0065] FIG. 8 illustrates details of the information maintained within the workflow service object (server side) 558, including a workflow name 600, workflow type 602, workflow category 604, and a connection object 608. Information in the workflow service object (server side) 558 is accessed using the methods of the WorkFlowService (client side) 506 and (server side) 508 classes. As discussed, in certain implementations, the workflow service object (client side 552a, b, c) does not include specific information, and instead comprises a proxy object for method calls from the WorkFlowService (client side) 506 class that are communicated to the workflow server 556 and implemented through the WorkFlowService (server side) 508 class against the workflow service object (server side) 558. There would be a separate instance of a workflow service object

(client side and server side) for each connection to the workflow engine 560, such that each workflow service object maintains information on only one connection. In alternative implementations, one service o ject may maintain multiple connection objects for different connections.

[0066] FIGs. 9, 10, and 11 illustrate logic implemented in the methods of the service classes (client side) 506, 510 and (server side) 508, 512. In certain implementations, the service classes are implemented in an object oriented programming language, such as Java, C++, etc., and the clients 550a, b, c, 570a, b, c and servers 556 and 576 would include platforms capable of executing the methods of the service classes. For instance, if the service classes are implemented in Java, then the clients 550a, b, c, 570a, b, c and servers 556, 576 would include Java Virtual Machine (JVM) code to execute the methods of the service classes.

[0067] FIG. 9 illustrates logic implemented in the WorkFlowService() and OtherService() methods to instantiate a service object in the service classes (client side) 506, 510 and (server side) 508, 512 to instantiate client side 552a, b, c, 572a, b, c and/or server side 558, 578 service objects, respectively. Control begins at block 650 where a method call is received to construct a service object. If (at block 652) the method call is at the client 550a, b, c, 570a, b, c, then the method call causes the clients 550a, b, c, 570a, b, c, to construct (at block 654) a client side service object 552a, b, c, 572a, b, c and communicate (at block 656) the method to the workflow server 556. If the method call was invoked at the server 556 (from the no branch of block 652) or from block 656, then the method call would cause the server 556, 576 to instantiate (at block 658) a server side service object 558, 578 and load (at block 660) the instantiated service object into memory. Once the service object is instantiated, the client applications 562, 584 can then issue method calls to access information from the engines 560, 582 using the instantiated service object.

[0068] FIG. 10 illustrates logic implemented in the connect() method to instantiate a service object in the service classes (client side) 506, 510 and (server side) 508, 512 to

construct connection object 608 (FIG. 8). Control begins at block 670 upon receiving a call to construct a connection object, including as parameters, a user name, service name, and user authentication information. If (at block 672) the call is client side, i.e., at the clients 550a, b, c, 570a, b, c, then the connect() method is passed to the server 556, 576 via the communication protocol 554a, b, c, 574a, b, c. From the no branch of block 672 or from block 674, the server 556, 576 executes (at block 676) the connect() method and calls the interface 564, 580 to transform the method to the native code of the engine 560, 582. The engine 560, 582 then attempts authentication (at block 678) for the user requesting access to the engine 560, 582 resources. If (at block 680) the user access attempt did not authenticate, then the engine 560, 582 returns (at block 682) a failure message. Otherwise, if authentication succeeded, then the engine 560, 582 instantiates (at block 684) a connection object 608 (FIG. 8) including authentication information for the requested service and user name. The connection object 608 is then stored (at block 686) in the service object 552a, b, c, 572a, b, c, and a connection handle addressing the connection object is returned to the caller to use in subsequent accesses of the engine 560, 572.

[0069] FIG. 11 illustrates logic implemented in the methods to access information on an engine in the service classes (client side) 506, 510 and (server side) 508, 512. Control begins at block 700 upon receiving a call to a method to obtain information for a named service, including a connection handle. For instance, the method may request the name of the service, type, category or resources at the engine 560, 582, e.g., work lists, workflow templates, workflows, etc. If (at block 702) the method is invoked on the client 550a, b, c, 570a, b, c, then the method is transmitted (at block 702) to the server 556, 576. If (at block 706) the method is requesting information maintained in the service object 558, 578, e.g., the category, type, connection, etc., then the information is accessed (at block 708) from the service object 558, 578 and returned to the caller. Otherwise, if the requested information is not maintained in the service object 558, 578, then the interface 564, 580 is called (at block 710) to transform the method into the native engine 560, 582

code. The engine 560, 582 uses the handle (at block 712) to access the connection object from the named service 558, 578 to authenticate the user requesting the information. If (at block 714) the information provided in the connection object addressed by the connection handle did not provide an active and authenticated connection, then a failure message is returned (at block 716). Otherwise, if the connection is authenticated, then the engine 560, 582 executes (at block 718) the method in the transformed native code, accesses the requested information, e.g., name of workflows, work lists, workflow templates, etc., and returns the requested information to the calling method.

[0070] With the described implementations, the service objects 558, 578 maintain information on one engine 560, 582 for different connections. Application 562 and 584 and users would utilize methods from the service classes 506, 508, 510, and 512 to access information on engine 560, 582 resources from the service objects 558, 578 or the engines 560, 582 using the connection object 608 information in the service objects 558, 578. The methods to access information from the service object may have originated from methods in the client side 506, 510 or server side 508, 512 of the class implementations. Further, the service objects 558 and 578 provide authentication services and information that the users and applications 562 and 584 may access using one connection handle. Once the applications 562, 584 or users obtain information on resources available for an engine 560, 582, the user or application may then access the services directly using methods and objects provided by the vendor for the service to access the particular service, such as the object oriented workflow class architecture described with respect to FIGs. 3, 4, and 5.

[0071] In the described implementations, the workflow 562 and other service 584 applications would include the methods and objects of the vendor implementations of the WorkFlowServices

classes 506, 508, 510, 512 to access the engine 560, 582 resources and information thereon. The applications 562 and 584 may include methods and objects from the service implementations from multiple vendors, thereby allowing the applications to access

resources from the services provided by different vendors. For instance, an application may include methods and objects to perform a search across data stores implementing data repositories from different vendors, wherein each vendor provides a service class implementation to enable access to the data repository. This allows for searches, data mining operations, and workflows to access data across multiple, heterogeneous content servers. In this way, developers may create applications 562 and 584 that are capable of accessing the services provided by different vendors. Moreover, one application may include methods to access different types of resources, such as methods to access workflow resources from one vendor and methods to access the resources of another service. Still further, one application may include methods and objects from different vendors to access the resources of one type of service as provided by different vendors.

[0072] The abstract classes 500, 502, and 504 provide base level methods and objects that must be included in all vendor implementations of the service class to provide a standard methodology as to how resources are accessed and how information on the service is provided in the service objects 558, 578. As discussed above, the abstract classes provide methods and objects concerning how to connect to an engine 560, 582, such as the connect() and disconnect() methods, and how to obtain information on available engine 560, 582 resources, such as the serviceType(), serviceName(), serviceCategory() methods, which apply to all types of services. The abstract workFlowService class 502 includes methods and objects to obtain information on the resources of the workflow services, such as the getDataStore(), listWorkLists(), listWorkFlows(), and listWorkFlowTemplates(). By requiring all participating vendors to implement the same abstract service classes, the code for different vendors utilize the same methods and objects to provide access to the resources and information.

Gathering and Distributing Service Information

[0073] In the above described implementations, the applications 562, 584 or users may request information on available engine 560, 582 resources through various method calls

in a service class implementation, such as listWorkflows, listWorkLists, and listWorkFlowtemplates to obtain information on workflows, work lists, and workflow templates. Further implementations provide techniques for delivering up-to-date information on resources available at the engines 560, 582 to application programs 562, 584 and users requesting such information using the methods of the service class implementations 506, 508, 510, 512.

[0074] In certain implementations, a collection object class is provided to transport information from the engines 560, 582 to a calling entity, such as a calling application 562, 584 or user on the client or server sides. The collection object class includes a collection object that maintains metadata on different resources in the engine 560, 582. For instance, the collection object may be used to maintain information on work lists, workflows, work templates, etc., in the workflow engine 560, 582. Each vendor wanting to make their engine 560, 582 available to users and applications 562, 584 in the federated system would implement the collection object class on both the server side and client side. For instance, the collection object class may be part of the service class implementations 506, 508, 510, 512 for each engine 560, 582. Thus, the collection object class may comprise an abstract class implemented by all vendors whose engine products are to be included in the system.

[0075] The collection object class may include the following methods used to manage collection objects.

Add: inserts metadata for a resource object, e.g., workflow, work list, work template, etc., in the engine 560, 582 into a collection object.

Delete: removes metadata for a resource object from the collection object.

Update: updates a workflow object.

Retrieve: retrieves the actual resource represented as metadata in the collection object.

Create an iterator: creates an index into the collection object referencing one metadata element in the collection object.

Move Iterator Next: moves the iterator to a next metadata element in the collection object.

Move Iterator Previous: moves the iterator to a previous metadata element in the collection object.

Set Iterator: sets the iterator to a specific metadata element in the collection object.

[0076] FIG. 12 illustrates an example of a collection object 750 including metadata elements 752a, b, c, d on workflow objects available at the workflow engine 560. The metadata may include the name of the workflow object and additional brief descriptive information. The collection object 750 can be instantiated in response to a call to the method listWorkFlows that returns metadata on all workflows at a particular workflow engine 560,the method listWorkFlowTempates that returns metadata on all workflow templates at one workflow engine 560, and to the method listWorkLists that returns metadata for each work list available at one workflow engine 560. Similarly, a collection object may be constructed to maintain metadata for resources at the other service engine 582, such as a content server, and include metadata on queries, searchable resources, etc.

[0077] FIGs. 13 and 14 illustrate logic to implement the methods in the WorkFlowService 506, 508 and OtherService 510, 512 classes. FIG. 13 illustrates logic implemented in methods to access lists of resources in the service classes 506, 508, 510, and 512, such as the listWorkFlows, listWorkLists, listWorkFlowTemplates, etc. Control begins at block 800 when the server 556, 576 receives a method to access a list of resources. If (at block 802) the call was initiated at a client 550a, b, c, 570a, b, c, then the method is passed (at block 804) to the server 556, 576 to execute. Whether the method is invoked from the server 556, 576 (the no branch of block 802) or the clients 556a, b, c, 570a, b, c (from block 804), the server 556, 576 calls (at block 806) the interface 564, 580 to transforms the method to the engine 560, 582 native code and transmits the native code to the engine 560, 582. In response, the engine 560, 582 would generate a list describing

the instances of a particular engine resource, e.g., work lists, workflows, workflow templates, etc.

[0078] Upon receiving (at block 808) a list of the engine resources 560, 582 from the engine 560, 582, the server 556 would instantiate (at block 810) a collection object 750. For each resource instance included in the received list, the server 556 would add (at block 812) a metadata element 752a, b, c, d to the collection object 750 (FIG. 12). The sever 556 would utilize the "Add" method in the collection object class to insert metadata into the collection object 750. Once the collection object 750 includes metadata element for each engine resource instance included in the list, the server 556, 576 returns (at block 814) the collection object 750 to the calling method, which may be local to the server 556, 576 or at a remote client 550a, b, c, 570a, b, c.

[0079] FIG. 14 illustrates the logic of blocks 830 to 838 implemented in the applications 562, 584 or by a user to access data in the received collection object 750. Upon receiving (at block 830) the collection object 750, an iterator is created (at bock 832) to point to a first metadata element 752a, b, c, d in the collection object 750. The user or application 562, 584 would then use (at block 834) the move methods to move the iterator through the collection object 750 to access and review metadata elements 752a, b, c, d. One or more metadata elements 752a, b, c, d may then be selected (at block 836) to access. For each selected metadata element, a retrieve method, from the collection object class, is issued (at block 838) to access the actual resource, e.g., workflow object, work list object, workflow template, etc., represented by the selected metadata element 752a, b, c, d.

[0080] Blocks 850 to 858 illustrate logic implemented in the collection object class to allow the server 556, 576 to process a call to the retrieve method. At block 850, the server 556, 576 receives a retrieve method from the collection object class. The server then determines (at block 852) the method in the server side service class 508, 512 that would retrieve the requested resource element, e.g., workflow object, work list, workflow template, etc. and generates (at block 854) engine 560, 582 native code to cause the

engine 560, 582 to implement the determined method and return the requested resource object. At block 856, the server 556 receives the returned resource object from the engine 560, 582 represented by the requested metadata element and returns the resource object to the calling method, which may be local to the server 556, 576 or remote at one of the clients 550a, b, c, 570a, b, c.

[0081] The described implementations provide a collection object class to provide information on resources available at an application program, such as the workflow engine 560. The actual resource objects can be very large in size, e.g., hundreds of megabytes. The collection object class allows a server to return metadata on available resources at an application program to a requesting user or application. The user or application may then use the collection object class methods to review the metadata on the resources in the collection object and retrieve one or more resource objects represented by metadata elements in the collection object. The resource object would then be retrieved from the engine and returned to the requesting user or application.

[0082] The described implementations provide just-in-time retrieval of data from an application, such as a workflow engine, on both the server and client side when the application needs data. Network performance is improved because only the minimal amount of information needed to satisfy the current level of request at the application is transferred to the application, such as metadata elements describing resource objects in the application, where each resource object may be very large in size. This aspect allows immediate transmittal of information on the resources in the engine. The application may then request the actual resource object represented by the metadata. In this way, unnecessary retrieval of large resource objects, and transfer of such data between the servers and clients, is avoided because only specifically requested large resource objects that the application needs are retrieved and transferred over the network. Unneeded copies of resource objects are not maintained at the client or servers, and left in the engine until specifically requested.

Additional Implementation Details

[0083] The preferred embodiments may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software or code. The term "article of manufacture" as used herein refers to code or logic implemented in a computer readable medium (e.g., magnetic storage medium (e.g., hard disk drives, floppy disks, tape, etc.), optical storage (CD-ROMs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, firmware, programmable logic, etc.). Code in the computer readable medium is accessed and executed by a processor. The code in which preferred embodiments are implemented may further be accessible through a transmission media or from a file server over a network. In such cases, the article of manufacture in which the code is implemented may comprise a transmission media, such as a network transmission line, wireless transmission media, signals propagating through space, radio waves, infrared signals, etc. Of course, those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present invention, and that the article of manufacture may comprise any information bearing medium known in the art.

[0084] The workflow client and server may be implemented within any vendor workflow program known in the art.

[0085] In the described implementations, the actions were implemented as Java methods. Alternatively, the actions may be implemented in any programming language known in the art.

[0086] In the described implementations, particular icons were used to represent different information in the workflow, such as work nodes, exit nodes, etc. However, any icon design may be used to represent the workflow components. Further, additional graphical representations may be provided for different types of work nodes, e.g., collection work nodes, assign value node, decision point node, etc.

[0087] In the described implementations, the class architecture is implemented as an object oriented class architecture. Alternatively, non-object oriented programming techniques may be used to implement the described class architecture.

[0088] The services 552a, b, c, and 572a, b, c, may each execute in a separate computer system comprised of one or more computer devices. Additionally, multiple services 552a, b, c, 572a, b, c may execute in a same computer system.

[0089] The foregoing description of the preferred embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

**MQSeries, IBM, and DB2 are registered trademarks of International Business Machines Corp., Java is a trademark of Sun Microsystems, Inc.